游易-程序主题分享合辑

知识管理部 等 2022.06.07 15:37 6220 321 184个资源

汇总从101至今的游易征稿文章合辑。

专题 游易程序第132期 阴影算法

目录

作者风采和作品合集
【程序·132期】游易·本期上榜作家
直接光下的阴影实践
一种高质量静态阴影压缩方案
游戏中的阴影实现介绍
月移疏影上东墙——简述阴影映射算法
【Seed-TA】全自动生成风格化脸部光照贴图工具
从基础的阴影贴图到PSM，LiSPSM

一种高质量静态阴影压缩方案

陈宇晖 2021.05.21 17:07 1259 22 3 查看原文

做过阴影的小伙伴都知道，普通的CSM在面对大世界场景中，也会有点压力，尤其是现在的手机平台上。像去年分享的原机端使用的是改良后的8层CSM，其中前四帧每帧更新，后四层跨帧轮流更新，这使得CPU和GPU的开销都大大增加。我们目的是降低开销的同时，还要保证高质量阴影的高精度，所以下面我要分享一种名为稀疏阴影树（Sparse Shadow Tree，简称SST）的静态阴影压缩技术。

关键词： 静态阴影压缩、大世界阴影技术、四叉树、平面拟合

稀疏阴影树（Sparse Shadow Tree）

一、概述

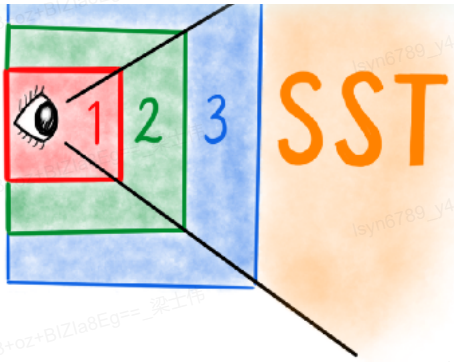
稀疏阴影树（Sparse Shadow Tree，下面简称SST）是一种静态阴影压缩方法，Treyarch在《使命召唤：黑色行的大型室外场景中应用了SST技术，并在图形学顶会SIGGRAPH 2016 TALK进行分享。在黑洞（邵国栋）的主NeoX引擎正在开发此项功能，后续的版本应该会放出，到时候欢迎尝试。目前已经和H69讨论该技术落地的细组应该已经提前实装了。

官方技术主页： https://research.activision.com/publications/archives/sparse-shadow-trees
ACM SIGGRAPH 2016 TALK Page： https://dl.acm.org/doi/10.1145/2897839.2927418

SST的主要流程概述如下，
- 对整个大地形的高精度shadowmap进行烘焙
- 用烘焙结果生成点云，并用四叉树进行阴影的场景管理
- 每层四叉树的点云数据都用最小二乘法进行压缩，如果拟合则保存，不能拟合则进入四叉树下一层
- 压缩完成生成SST资源文件
- 最后runtime，GPU解压SST文件还原阴影。

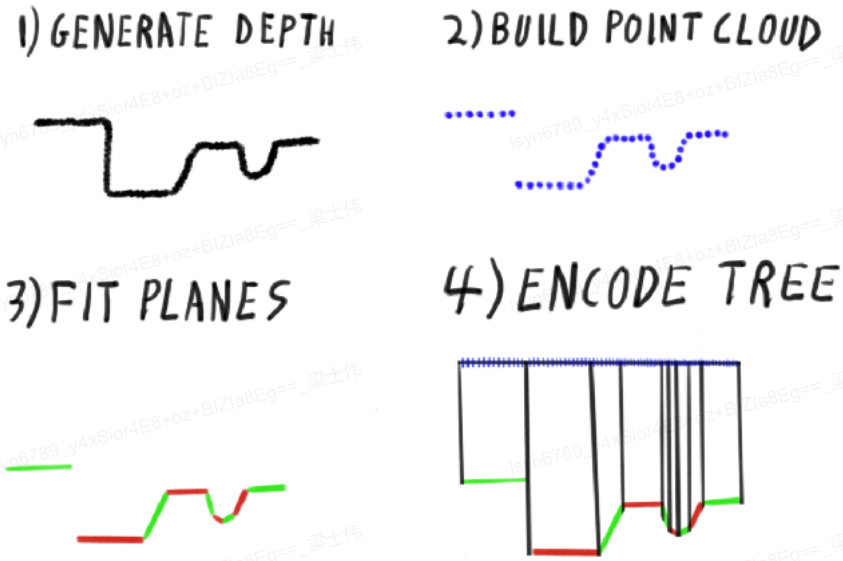
细心的小伙伴发现了，相比于正常ShadowMap流程，SST需要额外消耗解压的开销，但与此同时它省去了多张ShadowMapCaster的开销，而caster的开销（时间和空间都）是比解压多得多的，目前测试的一个NeoX中的简景，用512x512的精度shadowmap在pc上解压大概耗时0.02ms，基本符合预期。

但是这也就暴露出SST最大的缺点，即它是针对静态的阴影，它的压缩部分是一个离线算法，目前还没有办法做时，我自己测试初版本每个128x128的shadow map tile压缩大概需要0.01s左右。这是正常的，用闫令琪的话来trade-off。我们保证了阴影精度，降低了时间和空间的开销，但代价是要想办解决静态的问题。



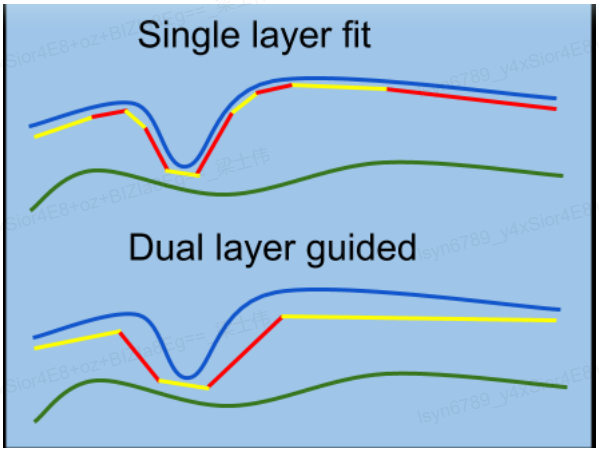
二、算法细节

我们再精简些，一句话概括这个算法的核心：其实就是ShadowMap的压缩和解压。而压缩和解压是相对的，我一种算法来压缩ShadowMap，再通过压缩算法的逆向解压还原出来。



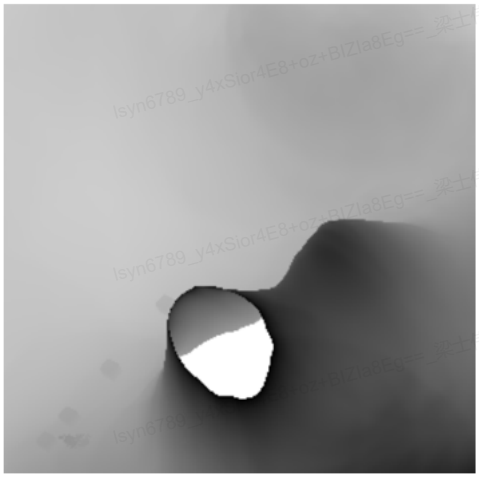
1.阴影烘焙

普通的ShadowMap是SST进行压缩的原材料，本质上就是进行shadow caster并收集的过程，项目组使用UE进
NeoX现在也实现了专门给SST使用的烘焙。
除了烘焙标准的ShadowMap，这里还需要烘焙**第二深度**，使用双层深度进行拟合，能有效改善自阴影，并提高
例。其实双深度ShadowMap在学术界很早就有提到，但因为开销等因素，实际应用场景并不常见。而我们的压
是离线的算法，应用双深度所带来的的损耗我们在游戏运行时是忽略不计的，正巧避开了开销的缺点，还给我们
很多好处。



第二深度的算法，在论文中没有提到烘焙的方式，我们可以使用深度剥离（Depth Peeling）的方法来做。这其
比较经典的用于绘制透明物体的算法，把多层的color和depth保存下来，最后根据alpha进行混合多层的结果，得
的感觉。

具体实现上，以往我们只渲染离相机最近的一层color和depth，现在我们要把这一层的depth保存下来，然后下
染的时候，如果遇到比这个depth更接近或一样近的就discard，这样就渲染出第二接近相机的depth。把这两层
下来，烘焙完成。下图为别人用Depth Peeling渲染头发，模拟头发的透明感，出处见下面的参考链接。更下面
烘焙的用于地形小场景的两层深度。



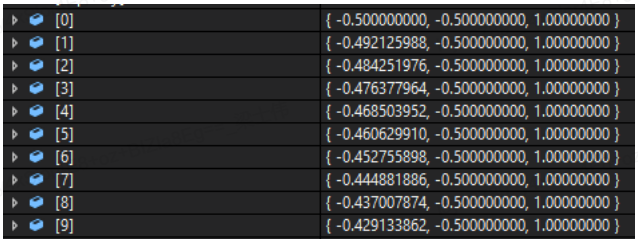
First Depth

Second Depth

2.生成点云

论文把上一步烘焙好的ShadowMap数据读取出来，然后按照像素级别构建点云数据。其中这里的点云不是使用
ShadowMap的数据，而是把整个ShadowMap拆成N个128x128大小的ShadowTile。
ShadowTile是SST进行压缩算法计算的基本单位，每个点云的数据量大小是固定的128x128个点。压缩的基本单
太大，会多出很多无意义的计算，同时也会加大四叉树的深度，会大大增加Decode的开销。

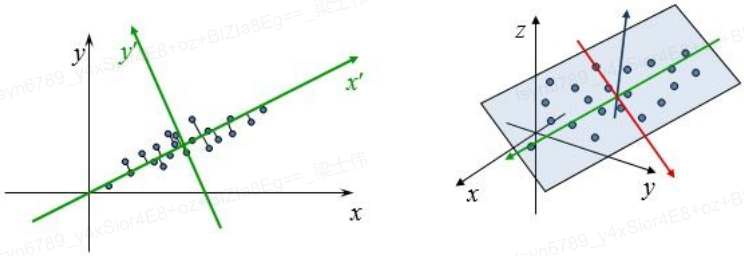
这里构建点云的时候，各个点的坐标要记得做对称处理，我使用的xy轴的区间范围是[-0.5，0.5]，z轴是深度范
经实验，没有做对称处理，压缩比例会少好几倍，因为拟合算的结果不够理想。下图是我构建的点云数据的示



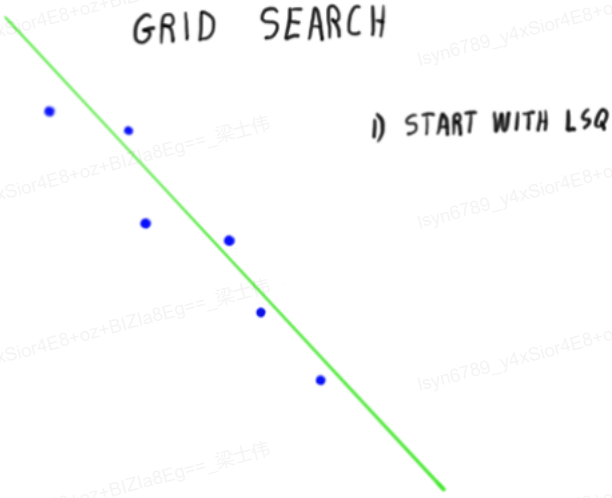
[0]	{ -0.500000000, -0.500000000, 1.00000000 }
[1]	{ -0.492125988, -0.500000000, 1.00000000 }
[2]	{ -0.484251976, -0.500000000, 1.00000000 }
[3]	{ -0.476377964, -0.500000000, 1.00000000 }
[4]	{ -0.468503952, -0.500000000, 1.00000000 }
[5]	{ -0.460629910, -0.500000000, 1.00000000 }
[6]	{ -0.452755898, -0.500000000, 1.00000000 }
[7]	{ -0.444881886, -0.500000000, 1.00000000 }
[8]	{ -0.437007874, -0.500000000, 1.00000000 }
[9]	{ -0.429133862, -0.500000000, 1.00000000 }

3.平面拟合

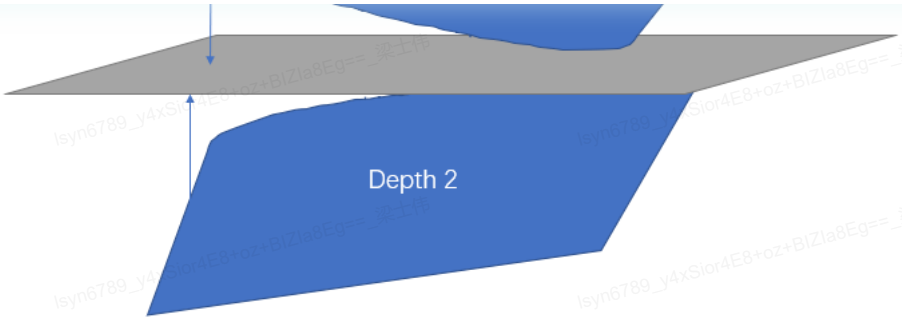
论文提到了两种方案，第一种是PCA算法，通过计算3x3的协方差矩阵，再求逆方差矩阵，最后通过power近似
找到法线和法线所代表的的拟合平面。但PCA收敛比较慢，导致阴影压缩速度很慢，故有了第二种方案。



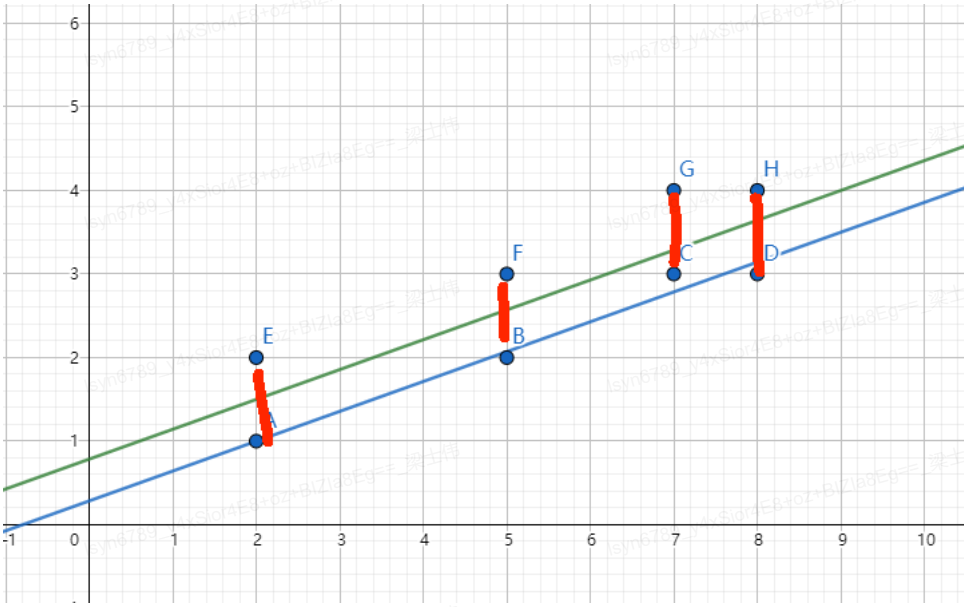
第二种是方法是最小二乘法Least squares （LSQ）。有解析解，我们通过线性代数的知识就能求出点云的拟合
详细推导见附件。我们以二维情况下为例，相当于求解一条直线使得离所有的点距离最短。



前面说过，我们要考虑双深度的优化，所以在平面拟合这部分我们也要考虑双层深度拟合的问题。一 还
么做到两个曲面拟合一个平面，甚至想过取两个平面的中点来做这个双深度的优化，这明显是偷懒的结果，最
效果很一般，压缩率提高很小，自阴影也没太大的改善。



后来还是使用最小二乘法应用到双层深度上。下面绿色线代表的这个拟合的平面。因此我们扩展到三维去计算
数，就能得到我们想要的效果。



其中ABCD属于一层深度，EFGH属于第二层深度。蓝线是单深度的拟合，绿线是双深度的拟合。

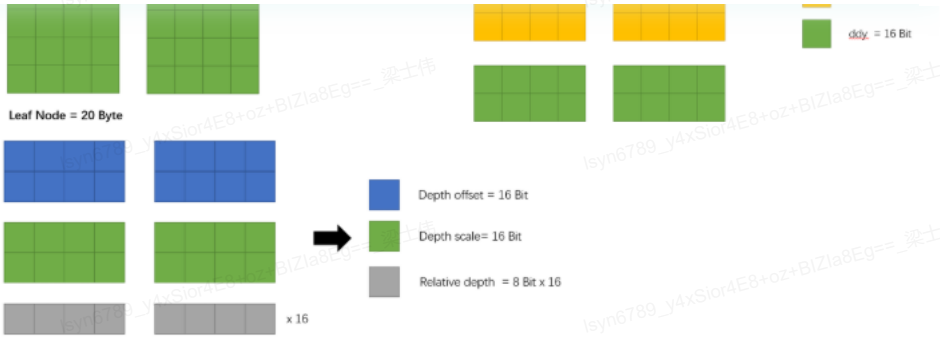
```
double atb[2] = {0};
double ata[2] = {0};

for( dvec3_t point : points )
{
    ata[0] += point.x * point.x;
    ata[1] += point.y * point.y;

    atb[0] += point.x * point.z;
    atb[1] += point.y * point.z;
}

double ddx = atb[0] / ata[0];
double ddy = atb[1] / ata[1];
```

```
double minZ = FLT_MAX;
double maxZ = -FLT_MAX;

for( dvec3_t point : points )
{
    double planeZ = ddx * point.x + ddy * point.y;
    double z = point.z - planeZ;
    minZ = min( minZ, z );
    maxZ = max( maxZ, z );
}

double error = maxZ - minZ;
double zOffset = maxZ;
```

有兴趣的可以手动推导一下，上面的代码可以转化成简单的线性代数计算。这里官方给的ppt的最小二乘法的计
有点小问题，zOffset计算出来不应该是最大值maxZ，我自己推导出来的结果应该是点云深度的平均值avgZ。

4.四叉树Encode

这里我的设计和论文里有点区别，我就讲我设计的四叉树节点存储结构好了。我把四叉树的节点分成三种类型，
前面2bit来标识区别。00 为父节点索引，10 为平面节点，11 叶子节点索引。总结的时候才发现论文里的叶子节
程度比我高，因为我多了一步叶子节点的索引，后续会再继续优化。



5.压缩流程

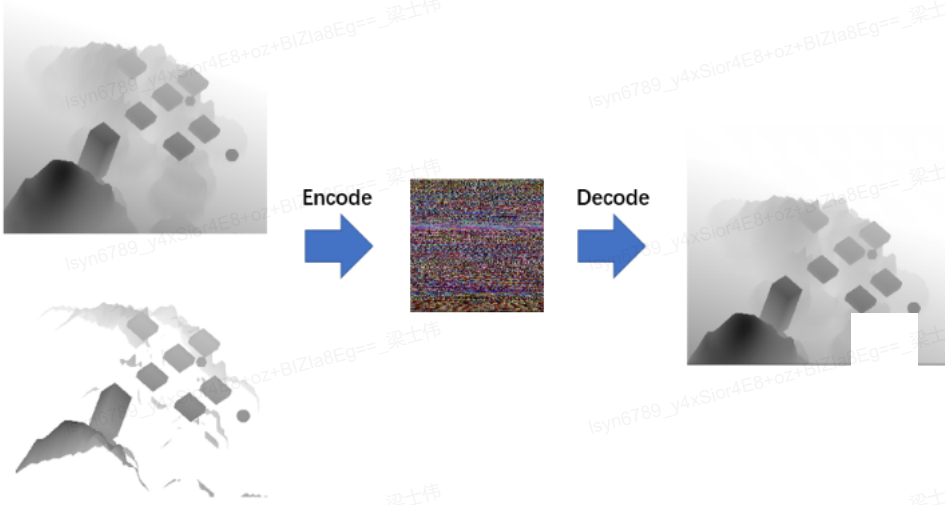
压缩时，根据LSQ计算平面是否拟合，如果拟合则保存平面数据，如果不拟合，则从128x128 -> 64x64->32x32->16x16->8x8->4x4进行宽度减半， 以此类推，直至4x4仍无法拟合，则把4x4的像素深度不再压缩，直接保存成点。四叉树的深度最大为6层。伪代码如下：

```
queue.push(pointclouds);
while(queue.size > 0)
{
    // 1.pop and get first pointcloud from queue

    // 2.try plane fit

    if(fit){
        // 3.save plane data(z_offset/ddx/ddy)
    }
    else{ // unfit
        if(reach max depth){
            // 4.save leaf data(index/z_offset/z_scale)
        }
        else{
            // 5.split into 4 child quad trees and push into queue
        }
    }
}
```

压缩完成后，保存成SST资源文件，供运行时Decode还原。

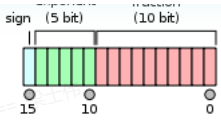
6.解压Decode

解压的原理就比较简单了，因为我们在压缩时已经在不拟合的父节点上建立了各种索引。把SST资源以Structur的形式送到shader，根据索引遍历四叉树，直至拟合的平面或者到达最深的叶子节点。而对于不支持StructureB手机会通过存成texture的形式来实现。



7.浮点数压缩

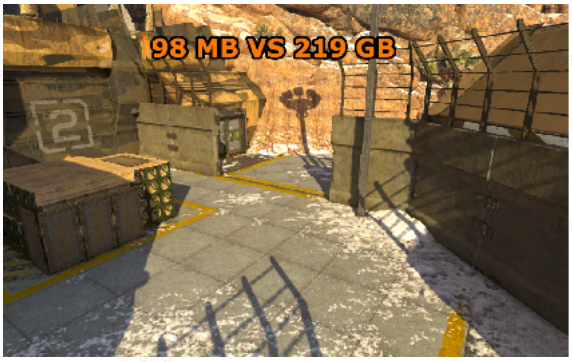
因为我把前两位用作节点类型的标记，所以需要对浮点数进行位数的压缩。主要是两种FP16和FP30。根据IEE浮点数的指数和尾数进行省略。



而FP30，我一开始是减去两位指数位，因为不想损失精度，但范围自己可以确定不需要太大的范围。后来在欢
瓠）同学的建议下，改成了去掉两位尾数位，原因是在写压缩和解压的时候会代码会工整很多，通过简单的移位
现。

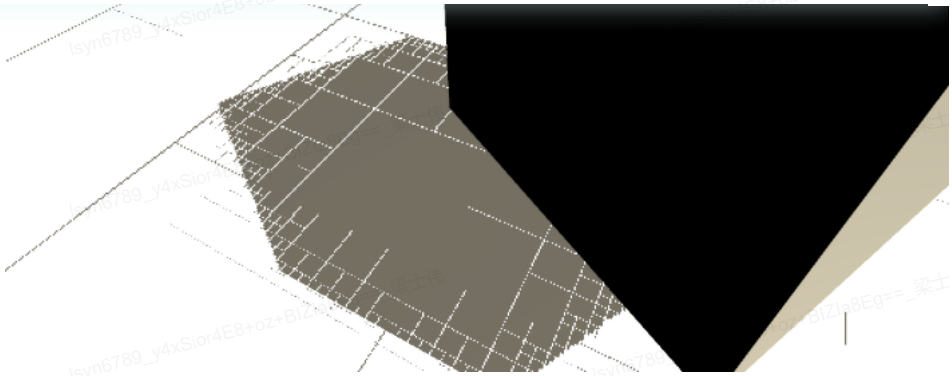
三、结果和展望

Treyarch在ppt中给出了压缩结果达到了千比一的惊人压缩比，原因是它的高精度ShadowMap实在太大了达到了
因此对于SST来说，通过四叉树的管理，可以压缩掉很多的非阴影部分。



实际上，我在NeoX的小场景中测试大概在百比一的水平，而一些像树叶这种高复杂难压缩的极端情况下，比如
叶，压缩比会降低到几十比一的情况。当然关于压缩率，还在不断优化中。

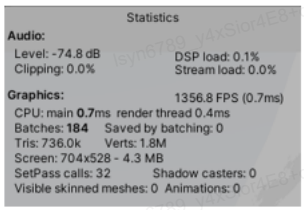
之前实现的时候，做了个调试模式，可视化阴影的四叉树。



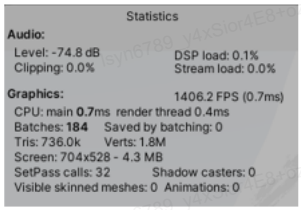
除此之外，还可以用在一些点光、聚光的静态场景，因为据我观察游戏场景中很多点光和聚光的应用场景其实都的。

关于性能，以我之前在Unity复现的结果展示如下。相同的森林大场景下，SST只需要大概0.7ms，CSM则大概用4.4ms左右。

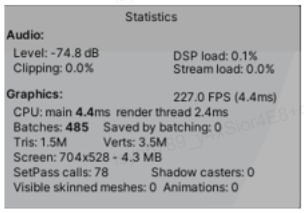
Forest + Cube (41trees)
shadow ratio 200+:1



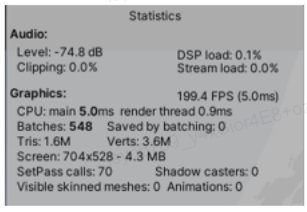
SST TEXTURE



SST buffer



4x CSM
4x 1024 x 1024



Shadow M
4096 x 409

后续正式集成到NeoX引擎后，会有更多平台更详细的测试结果和对比放出，也欢迎大家到时候使用并提建议。

参考

官方技术主页，部分技术细节示例图来源 https://research.activision.com/publications/archives/sparse-shadow-
ACM SIGGRAPH 2016 TALK https://dl.acm.org/doi/10.1145/2897839.2927418
PCA平面拟合示例图来源 https://zhuanlan.zhihu.com/p/56541912?utm_oi=988434103074709504
深度剥离示例图来源 https://zhuanlan.zhihu.com/p/127399447
浮点数压缩示例图来源 https://en.wikipedia.org/wiki/IEEE_754

还可以输入

(可添加1个视频+5张图片) 匿名

最热 最新

贾舟帆

3楼 感觉比csm scrolling复杂不少，并且还挺吃解压速率的

2021-11-14 17:30 回

贾舟帆 @黑洞(邵国栋) 大物体这点确实很坑。。。这是最大的硬伤

2022-03-18 10:12 回复

黑洞(邵国栋) PC上解压的速度和直接copy差不多，手机上略慢一点。scrolling的问题是，一些大物体cache是不友好
并且需要补画一些东西

2022-03-17 17:47 回复

刘主任(刘志斌)

2楼 用之前要解压为普通的shadowmap，最后只是省了包体么？
好像有方法是不解压 直接采样时候间接索引，然后连内存都省了？

2021-08-23 11:51 回

黑洞(邵国栋) 可以作为cache解压到shadow map，也可以直接运行时采样SST.

2022-03-17 17:48 回复

rs(任帅)

1楼 不知道是否有手机端的数据，我之前也实现了这个算法(类似的算法还有Voxel DAG），压缩率与你的结论类似，只有几
0：1，但是考虑到树结构解压对手机不太友好，加上带宽开销，我感觉不一定能击败CSM Scrolling之类的方法。

2021-06-15 14:47 回

贾舟帆 @rs(任帅)
后面GPU ZEN2的那个我看过了。。。我自己尝试复现后认为文章中的方案和实际游戏中的不一致，无法复
且附带的源码跑不起来且与文章不一致。

2022-03-18 10:17 复

rs(任帅) @贾舟帆
VoxelDAG理论上可以（其实就是预烘培多个太阳光角度），Fry Cry5有一个支持Time of Day的阴影缓存
PU Zen2, https://gpuzen.blogspot.com/2019/05/gpu-zen-2-parallax-corrected-cached.html)，不过该书没
电子版，所以尚不清楚原理。

2022-01-04 11:40 回复

贾舟帆 @rs(任帅) 请问一下这几个算法是不是都不太能在太阳光角度变化的情况下应用呀，暂时没想到太好的方案

137-项目研发工具... 136-调试及性能分... 135-人力手动生产 V... 134-包体相关的那... 133-游戏中的剧情... 132-阴影算法 131-非